Appendix A
C Program

```
#include <math.h>
#include <stdio.h>



struct term {
  char            name[100];
  char            cur_val[2000];
  struct equation   * epntr;
};

struct equation {
  struct term      * op;
  char            func[5];
  struct equation   *  nxt;
};

void solve (struct term * t_pntr) {

struct equation *epntr;
    epntr = t_pntr->epntr;
    do {
      if ( epntr == t_pntr->epntr && epntr->func[0] == '\0')
        sprintf(t_pntr->cur_val,"%s",epntr->op->cur_val);
      else if ( epntr == t_pntr->epntr && epntr->func[0] != '\0')
        sprintf(t_pntr->cur_val,"%s%s",epntr->op->cur_val,epntr->func);
      else if ( epntr != t_pntr->epntr && epntr->func[0] == '\0')
        sprintf(t_pntr->cur_val,"%s%s",t_pntr->cur_val,epntr->op->cur_val);
      else
        sprintf(t_pntr->cur_val,"%s%s%s",t_pntr->cur_val,epntr->op->cur_val,epntr-
>func);
      epntr = epntr->nxt;
    } while (epntr != NULL);
}

/* Computes parallel equations for:  A = B * C */
void main ()
{
struct term *areg;
struct term *breg;
struct term *creg;
struct term fb,dummy;
struct equation *epntr,*new_epntr;
int   gf_poly;


int   i,j,g[39];
int   mp[3];
int   start;
```

```
gf_poly  = 0x201b;

sprintf(dummy.name,"fb");
sprintf(dummy.cur_val,"DUMMY");
dummy.epntr   = NULL;


    areg = (struct term *)calloc(13,sizeof(struct term));
    breg = (struct term *)calloc(13,sizeof(struct term));
    creg = (struct term *)calloc(13,sizeof(struct term));

    /* create and initialize the data structure */

      sprintf(fb.name,"fb");
      sprintf(fb.cur_val,"X(12)");
      fb.epntr  = (struct equation *)calloc(1,sizeof(struct equation));
      fb.epntr->op = &areg[12];
      fb.epntr->func[0]  = '\0';
      fb.epntr->nxt = NULL;
      for (i=12;i>0;i--) {
        sprintf(areg[i].name,"A(%d)",i);
        sprintf(areg[i].cur_val,"A(%d)",i);
        if ((gf_poly >> i) & 1) {
          epntr = (struct equation *)calloc(4,sizeof(struct equation));
          areg[i].epntr   = epntr;
          new_epntr     = epntr;
          new_epntr++;

          epntr->op  = &areg[i-1];
          sprintf(epntr->func," xor ");
          epntr->nxt = new_epntr++;

          epntr++;
          epntr->op  = &fb;
          sprintf(epntr->func," xor ");
          epntr->nxt = new_epntr++;

          epntr++;
          epntr->op  = &breg[i];
          sprintf(epntr->func," and ");
          epntr->nxt = new_epntr;

          epntr++;
          epntr->op  = &creg[12];
          epntr->func[0]  = '\0';
          epntr->nxt = NULL;
        }
        else {

          epntr       = (struct equation *)calloc(3,sizeof(struct equation));
```

```
    areg[i].epntr = epntr;
    new_epntr     = epntr;
    new_epntr++;

    epntr->op  = &areg[i-1];
    sprintf(epntr->func," xor ");
    epntr->nxt = new_epntr++;

    epntr++;
    epntr->op  = &breg[i];
    sprintf(epntr->func," and ");
    epntr->nxt = new_epntr;

    epntr++;
    epntr->op  = &creg[12];
    epntr->func[0]  = '\0';
    epntr->nxt = NULL;
  }
}
sprintf(areg[0].name,"A(0)");
sprintf(areg[0].cur_val,"A(0)");
epntr        = (struct equation *)calloc(3,sizeof(struct equation));
areg[0].epntr  = epntr;
new_epntr      = epntr;
new_epntr++;

epntr->op  = &fb;
sprintf(epntr->func," xor ");
epntr->nxt = new_epntr++;

epntr++;
epntr->op  = &breg[i];
sprintf(epntr->func," and ");
epntr->nxt = new_epntr;

epntr++;
epntr->op  = &creg[12];
epntr->func[0]  = '\0';
epntr->nxt = NULL;

/* setup for creg */
for (i=12;i>0;i--) {
  sprintf(creg[i].name,"C(%d)",i);
  sprintf(creg[i].cur_val,"C(%d))",i);
  creg[i].epntr  = (struct equation *)calloc(1,sizeof(struct equation));
  creg[i].epntr->op   = &creg[i-1];
  creg[i].epntr->func[0] = '\0';
  creg[i].epntr->nxt  = NULL;
}
sprintf(creg[0].name,"C(0)");
```

```
    sprintf(creg[0].cur_val,"C(0))");
    creg[0].epntr      = (struct equation *)calloc(1,sizeof(struct equation));
    creg[0].epntr->op  = &dummy;
    creg[0].epntr->func[0] = '\0';
    creg[0].epntr->nxt  = NULL;

    /* setup for breg */
    for (i=12;i>0;i--) {
      sprintf(breg[i].name,"B(%d)",i);
      sprintf(breg[i].cur_val,"(B(%d)",i);
      breg[i].epntr      = (struct equation *)calloc(1,sizeof(struct equation));
      breg[i].epntr->op  = &breg[i-1];
      breg[i].epntr->func[0] = '\0';
      breg[i].epntr->nxt  = NULL;
    }
    sprintf(breg[0].name,"B(0)");
    sprintf(breg[0].cur_val,"(B(0)");
    breg[0].epntr      = (struct equation *)calloc(1,sizeof(struct equation));
    breg[0].epntr->op  = &dummy;
    breg[0].epntr->func[0] = '\0';
    breg[0].epntr->nxt  = NULL;

   /* execute the equations and update cur_values */
   for (j=0;j<5;j++) {
     solve(&fb);
     for (i=12;i>=0;i--) {
       solve(&areg[i]);
     }
     for (i=12;i>=0;i--)
       solve(&creg[i]);
   }
   /* write out the results */
   for (i=12;i>=0;i--)
     printf ("%s <= %s;\n",areg[i].name,areg[i].cur_val);
}
```

APPENDIX B

A(12) <= A(12) xor (B(0) and C(12)) xor A(11) xor (B(12) and C(12)) xor (B(1) and C(11)) xor (B(2) and C(10)) xor A(9) xor (B(10) and C(12)) xor (B(11) and C(11)) xor (B(12) and C(10)) xor (B(3) and C(9)) xor A(8) xor (B(9) and C(12)) xor (B(10) and C(11)) xor (B(11) and C(10)) xor (B(12) and C(9)) xor (B(4) and C(8)) xor (B(5) and C(7)) xor (B(6) and C(6)) xor (B(7) and C(5)) xor (B(8) and C(4)) xor (B(9) and C(3)) xor (B(10) and C(2)) xor (B(11) and C(1)) xor (B(12) and C(0));

A(11) <= A(11) xor (B(12) and C(12)) xor (B(0) and C(11)) xor A(10) xor (B(11) and C(12)) xor (B(12) and C(11)) xor (B(1) and C(10)) xor (B(2) and C(9)) xor A(8) xor (B(9) and C(12)) xor (B(10) and C(11)) xor (B(11) and C(10)) xor (B(12) and C(9)) xor (B(3) and C(8)) xor A(7) xor (B(8) and C(12)) xor (B(9) and C(11)) xor (B(10) and C(10)) xor (B(11) and C(9)) xor (B(12) and C(8)) xor (B(4) and C(7)) xor (B(5) and C(6)) xor (B(6) and C(5)) xor (B(7) and C(4)) xor (B(8) and C(3)) xor (B(9) and C(2)) xor (B(10) and C(1)) xor (B(11) and C(0));

A(10) <= A(10) xor (B(11) and C(12)) xor (B(12) and C(11)) xor (B(0) and C(10)) xor A(9) xor (B(10) and C(12)) xor (B(11) and C(11)) xor (B(12) and C(10)) xor (B(1) and C(9)) xor (B(2) and C(8)) xor A(7) xor (B(8) and C(12)) xor (B(9) and C(11)) xor (B(10) and C(10)) xor (B(11) and C(9)) xor (B(12) and C(8)) xor (B(3) and C(7)) xor A(6) xor (B(7) and C(12)) xor (B(8) and C(11)) xor (B(9) and C(10)) xor (B(10) and C(9)) xor (B(11) and C(8)) xor (B(12) and C(7)) xor (B(4) and C(6)) xor (B(5) and C(5)) xor (B(6) and C(4)) xor (B(7) and C(3)) xor (B(8) and C(2)) xor (B(9) and C(1)) xor (B(10) and C(0));

A(9) <= A(9) xor (B(10) and C(12)) xor (B(11) and C(11)) xor (B(12) and C(10)) xor (B(0) and C(9)) xor A(8) xor (B(9) and C(12)) xor (B(10) and C(11)) xor (B(11) and C(10)) xor (B(12) and C(9)) xor (B(1) and C(8)) xor (B(2) and C(7)) xor A(6) xor (B(7) and C(12)) xor (B(8) and C(11)) xor (B(9) and C(10)) xor (B(10) and C(9)) xor (B(11) and C(8)) xor (B(12) and C(7)) xor (B(3) and C(6)) xor A(5) xor (B(6) and C(12)) xor (B(7) and C(11)) xor (B(8) and C(10)) xor (B(9) and C(9)) xor (B(10) and C(8)) xor (B(11) and C(7)) xor (B(12) and C(6)) xor (B(4) and C(5)) xor (B(5) and C(4)) xor (B(6) and C(3)) xor (B(7) and C(2)) xor (B(8) and C(1)) xor (B(9) and C(0));

A(8) <= A(8) xor (B(9) and C(12)) xor (B(10) and C(11)) xor (B(11) and C(10)) xor (B(12) and C(9)) xor (B(0) and C(8)) xor A(7) xor (B(8) and C(12)) xor (B(9) and C(11)) xor (B(10) and C(10)) xor (B(11) and C(9)) xor (B(12) and C(8)) xor (B(1) and C(7)) xor (B(2) and C(6)) xor A(5) xor (B(6) and C(12)) xor (B(7) and C(11)) xor (B(8) and C(10)) xor (B(9) and C(9)) xor (B(10) and C(8)) xor (B(11) and C(7)) xor (B(12) and C(6)) xor (B(3) and C(5)) xor A(4) xor (B(5) and C(12)) xor (B(6) and C(11)) xor (B(7) and C(10)) xor (B(8) and C(9)) xor (B(9) and C(8)) xor (B(10) and C(7)) xor (B(11) and C(6)) xor (B(12) and C(5)) xor (B(4) and C(4)) xor (B(5) and C(3)) xor (B(6) and C(2)) xor (B(7) and C(1)) xor (B(8) and C(0));

A(7) <= A(7) xor (B(8) and C(12)) xor (B(9) and C(11)) xor (B(10) and C(10)) xor (B(11) and C(9)) xor (B(12) and C(8)) xor (B(0) and C(7)) xor A(6) xor (B(7) and C(12)) xor (B(8) and C(11)) xor (B(9) and C(10)) xor (B(10) and C(9)) xor (B(11) and C(8)) xor (B(12) and C(7)) xor (B(1) and C(6)) xor (B(2) and C(5)) xor A(4) xor (B(5) and C(12)) xor (B(6) and C(11)) xor (B(7) and C(10)) xor (B(8) and C(9)) xor (B(9) and C(8)) xor (B(10) and C(7)) xor (B(11) and C(6)) xor (B(12) and C(5)) xor

(B(3) and C(4)) xor A(3) xor A(12) xor (B(4) and C(12)) xor (B(5) and C(11)) xor (B(6) and C(10)) xor (B(7) and C(9)) xor (B(8) and C(8)) xor (B(9) and C(7)) xor (B(10) and C(6)) xor (B(11) and C(5)) xor (B(12) and C(4)) xor (B(4) and C(3)) xor (B(5) and C(2)) xor (B(6) and C(1)) xor (B(7) and C(0));

A(6) <= A(6) xor (B(7) and C(12)) xor (B(8) and C(11)) xor (B(9) and C(10)) xor (B(10) and C(9)) xor (B(11) and C(8)) xor (B(12) and C(7)) xor (B(0) and C(6)) xor A(5) xor (B(6) and C(12)) xor (B(7) and C(11)) xor (B(8) and C(10)) xor (B(9) and C(9)) xor (B(10) and C(8)) xor (B(11) and C(7)) xor (B(12) and C(6)) xor (B(1) and C(5)) xor (B(2) and C(4)) xor A(3) xor A(12) xor (B(4) and C(12)) xor (B(5) and C(11)) xor (B(6) and C(10)) xor (B(7) and C(9)) xor (B(8) and C(8)) xor (B(9) and C(7)) xor (B(10) and C(6)) xor (B(11) and C(5)) xor (B(12) and C(4)) xor (B(3) and C(3)) xor A(2) xor A(12) xor (B(3) and C(12)) xor A(11) xor (B(12) and C(12)) xor (B(4) and C(11)) xor (B(5) and C(10)) xor (B(6) and C(9)) xor (B(7) and C(8)) xor (B(8) and C(7)) xor (B(9) and C(6)) xor (B(10) and C(5)) xor (B(11) and C(4)) xor (B(12) and C(3)) xor (B(4) and C(2)) xor (B(5) and C(1)) xor (B(6) and C(0));

A(5) <= A(5) xor (B(6) and C(12)) xor (B(7) and C(11)) xor (B(8) and C(10)) xor (B(9) and C(9)) xor (B(10) and C(8)) xor (B(11) and C(7)) xor (B(12) and C(6)) xor (B(0) and C(5)) xor A(4) xor (B(5) and C(12)) xor (B(6) and C(11)) xor (B(7) and C(10)) xor (B(8) and C(9)) xor (B(9) and C(8)) xor (B(10) and C(7)) xor (B(11) and C(6)) xor (B(12) and C(5)) xor (B(1) and C(4)) xor (B(2) and C(3)) xor A(2) xor A(12) xor (B(3) and C(12)) xor A(11) xor (B(12) and C(12)) xor (B(4) and C(11)) xor (B(5) and C(10)) xor (B(6) and C(9)) xor (B(7) and C(8)) xor (B(8) and C(7)) xor (B(9) and C(6)) xor (B(10) and C(5)) xor (B(11) and C(4)) xor (B(12) and C(3)) xor (B(3) and C(2)) xor A(1) xor (B(2) and C(12)) xor A(11) xor (B(12) and C(12)) xor (B(3) and C(11)) xor A(10) xor (B(11) and C(12)) xor (B(12) and C(11)) xor (B(4) and C(10)) xor (B(5) and C(9)) xor (B(6) and C(8)) xor (B(7) and C(7)) xor (B(8) and C(6)) xor (B(9) and C(5)) xor (B(10) and C(4)) xor (B(11) and C(3)) xor (B(12) and C(2)) xor (B(4) and C(1)) xor (B(5) and C(0));

A(4) <= A(4) xor (B(5) and C(12)) xor (B(6) and C(11)) xor (B(7) and C(10)) xor (B(8) and C(9)) xor (B(9) and C(8)) xor (B(10) and C(7)) xor (B(11) and C(6)) xor (B(12) and C(5)) xor (B(0) and C(4)) xor A(3) xor A(12) xor (B(4) and C(12)) xor (B(5) and C(11)) xor (B(6) and C(10)) xor (B(7) and C(9)) xor (B(8) and C(8)) xor (B(9) and C(7)) xor (B(10) and C(6)) xor (B(11) and C(5)) xor (B(12) and C(4)) xor (B(1) and C(3)) xor (B(2) and C(2)) xor A(1) xor (B(2) and C(12)) xor A(11) xor (B(12) and C(12)) xor (B(3) and C(11)) xor A(10) xor (B(11) and C(12)) xor (B(12) and C(11)) xor (B(4) and C(10)) xor (B(5) and C(9)) xor (B(6) and C(8)) xor (B(7) and C(7)) xor (B(8) and C(6)) xor (B(9) and C(5)) xor (B(10) and C(4)) xor (B(11) and C(3)) xor (B(12) and C(2)) xor (B(3) and C(1)) xor A(0) xor A(12) xor (B(1) and C(12)) xor (B(2) and C(11)) xor A(10) xor (B(11) and C(12)) xor (B(12) and C(11)) xor (B(3) and C(10)) xor A(9) xor (B(10) and C(12)) xor (B(11) and C(11)) xor (B(12) and C(10)) xor (B(4) and C(9)) xor (B(5) and C(8)) xor (B(6) and C(7)) xor (B(7) and C(6)) xor (B(8) and C(5)) xor (B(9) and C(4)) xor (B(10) and C(3)) xor (B(11) and C(2)) xor (B(12) and C(1)) xor (B(4) and C(0));

A(3) <= A(3) xor A(12) xor (B(4) and C(12)) xor (B(5) and C(11)) xor (B(6) and C(10)) xor (B(7) and C(9)) xor (B(8) and C(8)) xor (B(9) and C(7)) xor (B(10) and C(6)) xor (B(11) and C(5)) xor (B(12) and C(4)) xor (B(0) and C(3)) xor A(2) xor

A(12) xor (B(3) and C(12)) xor A(11) xor (B(12) and C(12)) xor (B(4) and C(11)) xor (B(5) and C(10)) xor (B(6) and C(9)) xor (B(7) and C(8)) xor (B(8) and C(7)) xor (B(9) and C(6)) xor (B(10) and C(5)) xor (B(11) and C(4)) xor (B(12) and C(3)) xor (B(1) and C(2)) xor (B(2) and C(1)) xor A(0) xor A(12) xor (B(1) and C(12)) xor (B(2) and C(11)) xor A(10) xor (B(11) and C(12)) xor (B(12) and C(11)) xor (B(3) and C(10)) xor A(9) xor (B(10) and C(12)) xor (B(11) and C(11)) xor (B(12) and C(10)) xor (B(4) and C(9)) xor (B(5) and C(8)) xor (B(6) and C(7)) xor (B(7) and C(6)) xor (B(8) and C(5)) xor (B(9) and C(4)) xor (B(10) and C(3)) xor (B(11) and C(2)) xor (B(12) and C(1)) xor (B(3) and C(0));

A(2) <= A(2) xor A(12) xor (B(3) and C(12)) xor A(11) xor (B(12) and C(12)) xor (B(4) and C(11)) xor (B(5) and C(10)) xor (B(6) and C(9)) xor (B(7) and C(8)) xor (B(8) and C(7)) xor (B(9) and C(6)) xor (B(10) and C(5)) xor (B(11) and C(4)) xor (B(12) and C(3)) xor (B(0) and C(2)) xor A(1) xor (B(2) and C(12)) xor A(11) xor (B(12) and C(12)) xor (B(3) and C(11)) xor A(10) xor (B(11) and C(12)) xor (B(12) and C(11)) xor (B(4) and C(10)) xor (B(5) and C(9)) xor (B(6) and C(8)) xor (B(7) and C(7)) xor (B(8) and C(6)) xor (B(9) and C(5)) xor (B(10) and C(4)) xor (B(11) and C(3)) xor (B(12) and C(2)) xor (B(1) and C(1)) xor (B(2) and C(0));

A(1) <= A(1) xor (B(2) and C(12)) xor A(11) xor (B(12) and C(12)) xor (B(3) and C(11)) xor A(10) xor (B(11) and C(12)) xor (B(12) and C(11)) xor (B(4) and C(10)) xor (B(5) and C(9)) xor (B(6) and C(8)) xor (B(7) and C(7)) xor (B(8) and C(6)) xor (B(9) and C(5)) xor (B(10) and C(4)) xor (B(11) and C(3)) xor (B(12) and C(2)) xor (B(0) and C(1)) xor A(0) xor A(12) xor (B(1) and C(12)) xor (B(2) and C(11)) xor A(10) xor (B(11) and C(12)) xor (B(12) and C(11)) xor (B(3) and C(10)) xor A(9) xor (B(10) and C(12)) xor (B(11) and C(11)) xor (B(12) and C(10)) xor (B(4) and C(9)) xor (B(5) and C(8)) xor (B(6) and C(7)) xor (B(7) and C(6)) xor (B(8) and C(5)) xor (B(9) and C(4)) xor (B(10) and C(3)) xor (B(11) and C(2)) xor (B(12) and C(1)) xor (B(1) and C(0));

A(0) <= A(0) xor A(12) xor (B(1) and C(12)) xor (B(2) and C(11)) xor A(10) xor (B(11) and C(12)) xor (B(12) and C(11)) xor (B(3) and C(10)) xor A(9) xor (B(10) and C(12)) xor (B(11) and C(11)) xor (B(12) and C(10)) xor (B(4) and C(9)) xor (B(5) and C(8)) xor (B(6) and C(7)) xor (B(7) and C(6)) xor (B(8) and C(5)) xor (B(9) and C(4)) xor (B(10) and C(3)) xor (B(11) and C(2)) xor (B(12) and C(1)) xor (B(0) and C(0));

APPENDIX C

Appendix C
Sample VHDL Code

```
--
-- VHDL Entity fec.fdec_gfu.symbol
--
-- Generated by Mentor Graphics' Renoir(TM) 98.3 (Build 84)
--
LIBRARY ieee ;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY fdec_gfu IS
  PORT(
    ASEL     : in    std_logic_vector (1 downto 0) ;
    C1       : in    std_logic_vector (12 downto 0) ;
    CLK_78   : in    std_logic  ;
    D        : in    std_logic_vector (12 downto 0) ;
    MSEL     : in    std_logic_vector (2 downto 0) ;
    Q        : in    std_logic_vector (12 downto 0) ;
    RSTB     : in    std_logic  ;
    S        : in    std_logic_vector (12 downto 0) ;
    SIGMA_D  : in    std_logic_vector (12 downto 0) ;
    SIGMA_SEL : in   std_logic_vector (2 downto 0) ;
    GFU_Z    : out   std_logic  ;
    GF_OUT   : out   std_logic_vector (12 downto 0)
  );

-- Declarations

END fdec_gfu ;
--
-- VHDL Architecture fec.fdec_gfu.rtl
--
-- Generated by Mentor Graphics' Renoir(TM) 98.3 (Build 84)
--
architecture rtl of fdec_gfu is
  signal A          : std_logic_vector(12 downto 0);
  signal M_OP       : std_logic_vector(12 downto 0);
  signal A_OP       : std_logic_vector(12 downto 0);
  signal SIGMA      : std_logic_vector(12 downto 0);
  constant ZERO     : std_logic_vector(12 downto 0) := (others => '0');
  constant ONE      : std_logic_vector(12 downto 0) := "0000000000001";

begin


-- SIGMA Register plus input mux
sigma_p : process (CLK_78,RSTB)
begin
  if (RSTB = '0') then
    SIGMA <= (others => '0');
  elsif rising_edge(CLK_78) then
```

```
    case SIGMA_SEL is
      when "000"  => SIGMA <= SIGMA;
      when "001"  => SIGMA <= ZERO;
      when "010"  => SIGMA <= ONE;
      when "011"  => SIGMA <= SIGMA_D;
      when "100"  => SIGMA <= A;
      when others => SIGMA <= SIGMA;
    end case;
  end if;
end process;

-- Multipy operand register
m_op_p : process (CLK_78,RSTB)
begin
  if (RSTB = '0') then
    M_OP <= (others => '0');
  elsif rising_edge(CLK_78) then
    case MSEL is
      when "000" =>  M_OP <= ZERO;
      when "001" =>  M_OP <= ONE;
      when "010" =>  M_OP <= C1;
      when "011" =>  M_OP <= ONE;
      when "100" =>  M_OP <= S;
      when "101" =>  M_OP <= Q;
      when "110" =>  M_OP <= D;
      when others => M_OP <= M_OP;
    end case;
  end if;
end process;

-- Add operand register
a_op_p : process (CLK_78,RSTB)
begin
  if (RSTB = '0') then
    A_OP <= (others => '0');
  elsif rising_edge(CLK_78) then
    if (ASEL = "01") then
      A_OP <= D;
    elsif (ASEL = "10") then
      A_OP <= A;
    else
      A_OP <= ZERO;
    end if;
  end if;
end process;

-- Performs a galois field multiply/accumulate :
--  A <= (SIGMA * M_OP) + A_OP
-- SIGMA is assigned to the B variable and M_OP is assigned to
-- the C variable in order to keep the equations breif.
```

```
mult_add_p : process (SIGMA,M_OP,A_OP)
  variable B,C : std_logic_vector(12 downto 0);
begin
  B := SIGMA;
  C := M_OP;
A(12) <= ((B(0) and C(12)) xor (B(12) and C(12)) xor (B(1) and C(11)) xor (B(2) and
C(10)) xor (B(10) and C(12)) xor
      (B(11) and C(11)) xor (B(12) and C(10)) xor (B(3) and C(9)) xor (B(9) and
C(12)) xor (B(10) and C(11)) xor
      (B(11) and C(10)) xor (B(12) and C(9)) xor (B(4) and C(8)) xor (B(5) and C(7))
xor (B(6) and C(6)) xor
      (B(7) and C(5)) xor (B(8) and C(4)) xor (B(9) and C(3)) xor (B(10) and C(2))
xor (B(11) and C(1)) xor
      (B(12) and C(0)) ) xor A_OP(12);

A(11) <= ((B(12) and C(12)) xor (B(0) and C(11)) xor (B(11) and C(12)) xor (B(12)
and C(11)) xor (B(1) and C(10)) xor
      (B(2) and C(9)) xor (B(9) and C(12)) xor (B(10) and C(11)) xor (B(11) and
C(10)) xor (B(12) and C(9)) xor
      (B(3) and C(8)) xor (B(8) and C(12)) xor (B(9) and C(11)) xor (B(10) and
C(10)) xor (B(11) and C(9)) xor
      (B(12) and C(8)) xor (B(4) and C(7)) xor (B(5) and C(6)) xor (B(6) and C(5))
xor (B(7) and C(4)) xor
      (B(8) and C(3)) xor (B(9) and C(2)) xor (B(10) and C(1)) xor (B(11) and C(0)) )
xor A_OP(11);

A(10) <= ((B(11) and C(12)) xor (B(12) and C(11)) xor (B(0) and C(10)) xor (B(10)
and C(12)) xor (B(11) and C(11)) xor
      (B(12) and C(10)) xor (B(1) and C(9)) xor (B(2) and C(8)) xor (B(8) and C(12))
xor (B(9) and C(11)) xor
      (B(10) and C(10)) xor (B(11) and C(9)) xor (B(12) and C(8)) xor (B(3) and
C(7)) xor (B(7) and C(12)) xor
      (B(8) and C(11)) xor (B(9) and C(10)) xor (B(10) and C(9)) xor (B(11) and
C(8)) xor (B(12) and C(7)) xor
      (B(4) and C(6)) xor (B(5) and C(5)) xor (B(6) and C(4)) xor (B(7) and C(3)) xor
(B(8) and C(2)) xor
      (B(9) and C(1)) xor (B(10) and C(0)) ) xor A_OP(10);

A(9) <= ((B(10) and C(12)) xor (B(11) and C(11)) xor (B(12) and C(10)) xor (B(0)
and C(9)) xor (B(9) and C(12)) xor
      (B(10) and C(11)) xor (B(11) and C(10)) xor (B(12) and C(9)) xor (B(1) and
C(8)) xor (B(2) and C(7)) xor
      (B(7) and C(12)) xor (B(8) and C(11)) xor (B(9) and C(10)) xor (B(10) and
C(9)) xor (B(11) and C(8)) xor
      (B(12) and C(7)) xor (B(3) and C(6)) xor (B(6) and C(12)) xor (B(7) and C(11))
xor (B(8) and C(10)) xor
      (B(9) and C(9)) xor (B(10) and C(8)) xor (B(11) and C(7)) xor (B(12) and C(6))
xor (B(4) and C(5)) xor
      (B(5) and C(4)) xor (B(6) and C(3)) xor (B(7) and C(2)) xor (B(8) and C(1)) xor
(B(9) and C(0)) ) xor A_OP(9);
```

```
A(8) <= ((B(9) and C(12)) xor (B(10) and C(11)) xor (B(11) and C(10)) xor (B(12)
and C(9)) xor (B(0) and C(8)) xor
      (B(8) and C(12)) xor (B(9) and C(11)) xor (B(10) and C(10)) xor (B(11) and
C(9)) xor (B(12) and C(8)) xor
      (B(1) and C(7)) xor (B(2) and C(6)) xor (B(6) and C(12)) xor (B(7) and C(11))
xor (B(8) and C(10)) xor
      (B(9) and C(9)) xor (B(10) and C(8)) xor (B(11) and C(7)) xor (B(12) and C(6))
xor (B(3) and C(5)) xor
      (B(5) and C(12)) xor (B(6) and C(11)) xor (B(7) and C(10)) xor (B(8) and C(9))
xor (B(9) and C(8)) xor
      (B(10) and C(7)) xor (B(11) and C(6)) xor (B(12) and C(5)) xor (B(4) and C(4))
xor (B(5) and C(3)) xor
      (B(6) and C(2)) xor (B(7) and C(1)) xor (B(8) and C(0)) ) xor A_OP(8);

A(7) <= ((B(8) and C(12)) xor (B(9) and C(11)) xor (B(10) and C(10)) xor (B(11) and
C(9)) xor (B(12) and C(8)) xor
      (B(0) and C(7)) xor (B(7) and C(12)) xor (B(8) and C(11)) xor (B(9) and C(10))
xor (B(10) and C(9)) xor
      (B(11) and C(8)) xor (B(12) and C(7)) xor (B(1) and C(6)) xor (B(2) and C(5))
xor (B(5) and C(12)) xor
      (B(6) and C(11)) xor (B(7) and C(10)) xor (B(8) and C(9)) xor (B(9) and C(8))
xor (B(10) and C(7)) xor
      (B(11) and C(6)) xor (B(12) and C(5)) xor (B(3) and C(4)) xor (B(4) and C(12))
xor (B(5) and C(11)) xor
      (B(6) and C(10)) xor (B(7) and C(9)) xor (B(8) and C(8)) xor (B(9) and C(7))
xor (B(10) and C(6)) xor
      (B(11) and C(5)) xor (B(12) and C(4)) xor (B(4) and C(3)) xor (B(5) and C(2))
xor (B(6) and C(1)) xor
      (B(7) and C(0)) ) xor A_OP(7);

A(6) <= ((B(7) and C(12)) xor (B(8) and C(11)) xor (B(9) and C(10)) xor (B(10) and
C(9)) xor (B(11) and C(8)) xor
      (B(12) and C(7)) xor (B(0) and C(6)) xor (B(6) and C(12)) xor (B(7) and C(11))
xor (B(8) and C(10)) xor
      (B(9) and C(9)) xor (B(10) and C(8)) xor (B(11) and C(7)) xor (B(12) and C(6))
xor (B(1) and C(5)) xor
      (B(2) and C(4)) xor (B(4) and C(12)) xor (B(5) and C(11)) xor (B(6) and C(10))
xor (B(7) and C(9)) xor
      (B(8) and C(8)) xor (B(9) and C(7)) xor (B(10) and C(6)) xor (B(11) and C(5))
xor (B(12) and C(4)) xor
      (B(3) and C(3)) xor (B(3) and C(12)) xor (B(12) and C(12)) xor (B(4) and C(11))
xor (B(5) and C(10)) xor
      (B(6) and C(9)) xor (B(7) and C(8)) xor (B(8) and C(7)) xor (B(9) and C(6)) xor
(B(10) and C(5)) xor
      (B(11) and C(4)) xor (B(12) and C(3)) xor (B(4) and C(2)) xor (B(5) and C(1))
xor (B(6) and C(0)) ) xor A_OP(6);

A(5) <= ((B(6) and C(12)) xor (B(7) and C(11)) xor (B(8) and C(10)) xor (B(9) and
C(9)) xor (B(10) and C(8)) xor
```

(B(11) and C(7)) xor (B(12) and C(6)) xor (B(0) and C(5)) xor (B(5) and C(12)) xor (B(6) and C(11)) xor
(B(7) and C(10)) xor (B(8) and C(9)) xor (B(9) and C(8)) xor (B(10) and C(7)) xor (B(11) and C(6)) xor
(B(12) and C(5)) xor (B(1) and C(4)) xor (B(2) and C(3)) xor (B(3) and C(12)) xor (B(12) and C(12)) xor
(B(4) and C(11)) xor (B(5) and C(10)) xor (B(6) and C(9)) xor (B(7) and C(8)) xor (B(8) and C(7)) xor
(B(9) and C(6)) xor (B(10) and C(5)) xor (B(11) and C(4)) xor (B(12) and C(3)) xor (B(3) and C(2)) xor
(B(2) and C(12)) xor (B(12) and C(12)) xor (B(3) and C(11)) xor (B(11) and C(12)) xor (B(12) and C(11)) xor
(B(4) and C(10)) xor (B(5) and C(9)) xor (B(6) and C(8)) xor (B(7) and C(7)) xor (B(8) and C(6)) xor
(B(9) and C(5)) xor (B(10) and C(4)) xor (B(11) and C(3)) xor (B(12) and C(2)) xor
(B(4) and C(1)) xor (B(5) and C(0))) xor A_OP(5);

A(4) <= ((B(5) and C(12)) xor (B(6) and C(11)) xor (B(7) and C(10)) xor (B(8) and C(9)) xor (B(9) and C(8)) xor
(B(10) and C(7)) xor (B(11) and C(6)) xor (B(12) and C(5)) xor (B(0) and C(4)) xor (B(4) and C(12)) xor
(B(5) and C(11)) xor (B(6) and C(10)) xor (B(7) and C(9)) xor (B(8) and C(8)) xor (B(9) and C(7)) xor
(B(10) and C(6)) xor (B(11) and C(5)) xor (B(12) and C(4)) xor (B(1) and C(3)) xor (B(2) and C(2)) xor
(B(2) and C(12)) xor (B(12) and C(12)) xor (B(3) and C(11)) xor (B(11) and C(12)) xor (B(12) and C(11)) xor
(B(4) and C(10)) xor (B(5) and C(9)) xor (B(6) and C(8)) xor (B(7) and C(7)) xor (B(8) and C(6)) xor
(B(9) and C(5)) xor (B(10) and C(4)) xor (B(11) and C(3)) xor (B(12) and C(2)) xor (B(3) and C(1)) xor
(B(1) and C(12)) xor (B(2) and C(11)) xor (B(11) and C(12)) xor (B(12) and C(11)) xor (B(3) and C(10)) xor
(B(10) and C(12)) xor (B(11) and C(11)) xor (B(12) and C(10)) xor (B(4) and C(9)) xor (B(5) and C(8)) xor
(B(6) and C(7)) xor (B(7) and C(6)) xor (B(8) and C(5)) xor (B(9) and C(4)) xor (B(10) and C(3)) xor
(B(11) and C(2)) xor (B(12) and C(1)) xor (B(4) and C(0))) xor A_OP(4);

A(3) <= ((B(4) and C(12)) xor (B(5) and C(11)) xor (B(6) and C(10)) xor (B(7) and C(9)) xor (B(8) and C(8)) xor
(B(9) and C(7)) xor (B(10) and C(6)) xor (B(11) and C(5)) xor (B(12) and C(4)) xor (B(0) and C(3)) xor
(B(3) and C(12)) xor (B(12) and C(12)) xor (B(4) and C(11)) xor (B(5) and C(10)) xor (B(6) and C(9)) xor
(B(7) and C(8)) xor (B(8) and C(7)) xor (B(9) and C(6)) xor (B(10) and C(5)) xor (B(11) and C(4)) xor
(B(12) and C(3)) xor (B(1) and C(2)) xor (B(2) and C(1)) xor (B(1) and C(12)) xor (B(2) and C(11)) xor

```
      (B(11) and C(12)) xor (B(12) and C(11)) xor (B(3) and C(10)) xor (B(10) and
C(12)) xor (B(11) and C(11)) xor
      (B(12) and C(10)) xor (B(4) and C(9)) xor (B(5) and C(8)) xor (B(6) and C(7))
xor (B(7) and C(6)) xor
      (B(8) and C(5)) xor (B(9) and C(4)) xor (B(10) and C(3)) xor (B(11) and C(2))
xor (B(12) and C(1)) xor
      (B(3) and C(0)) ) xor A_OP(3);

A(2) <= ((B(3) and C(12)) xor (B(12) and C(12)) xor (B(4) and C(11)) xor (B(5) and
C(10)) xor (B(6) and C(9)) xor
      (B(7) and C(8)) xor (B(8) and C(7)) xor (B(9) and C(6)) xor (B(10) and C(5))
xor (B(11) and C(4)) xor
      (B(12) and C(3)) xor (B(0) and C(2)) xor (B(2) and C(12)) xor (B(12) and C(12))
xor (B(3) and C(11)) xor
      (B(11) and C(12)) xor (B(12) and C(11)) xor (B(4) and C(10)) xor (B(5) and
C(9)) xor (B(6) and C(8)) xor
      (B(7) and C(7)) xor (B(8) and C(6)) xor (B(9) and C(5)) xor (B(10) and C(4))
xor (B(11) and C(3)) xor
      (B(12) and C(2)) xor (B(1) and C(1)) xor (B(2) and C(0)) ) xor A_OP(2);

A(1) <= ((B(2) and C(12)) xor (B(12) and C(12)) xor (B(3) and C(11)) xor (B(11) and
C(12)) xor (B(12) and C(11)) xor
      (B(4) and C(10)) xor (B(5) and C(9)) xor (B(6) and C(8)) xor (B(7) and C(7))
xor (B(8) and C(6)) xor
      (B(9) and C(5)) xor (B(10) and C(4)) xor (B(11) and C(3)) xor (B(12) and C(2))
xor (B(0) and C(1)) xor
      (B(1) and C(12)) xor (B(2) and C(11)) xor (B(11) and C(12)) xor (B(12) and
C(11)) xor (B(3) and C(10)) xor
      (B(10) and C(12)) xor (B(11) and C(11)) xor (B(12) and C(10)) xor (B(4) and
C(9)) xor (B(5) and C(8)) xor
      (B(6) and C(7)) xor (B(7) and C(6)) xor (B(8) and C(5)) xor (B(9) and C(4)) xor
(B(10) and C(3)) xor
      (B(11) and C(2)) xor (B(12) and C(1)) xor (B(1) and C(0)) ) xor A_OP(1);

A(0) <= ((B(1) and C(12)) xor (B(2) and C(11)) xor (B(11) and C(12)) xor (B(12) and
C(11)) xor (B(3) and C(10)) xor
      (B(10) and C(12)) xor (B(11) and C(11)) xor (B(12) and C(10)) xor (B(4) and
C(9)) xor (B(5) and C(8)) xor
      (B(6) and C(7)) xor (B(7) and C(6)) xor (B(8) and C(5)) xor (B(9) and C(4)) xor
(B(10) and C(3)) xor
      (B(11) and C(2)) xor (B(12) and C(1)) xor (B(0) and C(0)) ) xor A_OP(0);
end process;

-- zero flag
zflag_p : process (CLK_78,RSTB)
begin
  if (RSTB = '0') then
    GFU_Z <= '0';
  elsif rising_edge(CLK_78) then
    if (A(0) = '1' or A(1) = '1' or A(2) = '1' or A(3) = '1' or A(4) = '1' or
```

```
        A(5) = '1' or A(6) = '1' or A(7) = '1' or A(8) = '1' or A(9) = '1' or
        A(10) = '1' or A(11) = '1' or A(12) = '1') then
        GFU_Z <= '0';
      else
        GFU_Z <= '1';
      end if;
    end if;
end process;

-- gf unit output register
gf_out_p : process (CLK_78,RSTB)
begin
  if (RSTB = '0') then
    GF_OUT <= (others => '0');
  elsif rising_edge(CLK_78) then
    GF_OUT <= A;
  end if;
end process;

end rtl;
```